

Fault Diagnosis with Dynamic Observers*

Franck Cassez[†]
CNRS, IRCCyN Laboratory
1 rue de la Noë
BP 92101
44321 Nantes Cedex 3
France

Email: franck.cassez@cnrs.irccyn.fr.

Stavros Tripakis
Cadence Research Laboratories
2150 Shattuck Avenue, 10th floor
Berkeley, CA, 94704
USA
and
CNRS, Verimag Laboratory
Centre Equation
2, avenue de Vignate, 38610 Gières
France
Email: tripakis@cadence.com.

Abstract—In this paper, we review some recent results about the use of dynamic observers for fault diagnosis of discrete event systems. Fault diagnosis consists in synthesizing a diagnoser that observes a given plant and identifies faults in the plant as soon as possible after their occurrence. Existing literature on this problem has considered the case of fixed static observers, where the set of observable events is fixed and does not change during execution of the system. In this paper, we consider dynamic observers: an observer can “switch” sensors on or off, thus dynamically changing the set of events it wishes to observe. It is known that checking diagnosability (i.e., whether a given observer is capable of identifying faults) can be solved in polynomial time for static observers, and we show that the same is true for dynamic ones. We also solve the problem of dynamic observers’ synthesis and prove that a most permissive observer can be computed in doubly exponential time, using a game-theoretic approach. We further investigate optimization problems for dynamic observers and define a notion of cost of an observer.

I. INTRODUCTION

A. Monitoring, Testing, Fault Diagnosis and Control

Many problems concerning the monitoring, testing, fault diagnosis and control of discrete event systems (DES) can be formalized using finite automata over a set of *observable* events Σ , plus a set of *unobservable* events [3], [4]. The invisible actions can often be represented by a single unobservable event ε . Given a finite automaton over $\Sigma \cup \{\varepsilon\}$ which is a model of a *plant* (to be monitored, tested, diagnosed or controlled) and an *objective* (good behaviours, what to test for, faulty behaviours, control objective) we want to check if a monitor/tester/diagnoser/controller exists that achieves the objective, and if possible to synthesize one automatically.

The usual assumption in this setting is that the set of observable events is fixed (and this in turn, determines the set of unobservable events as well). Observing an event usually requires some detection mechanism, i.e., a *sensor* of some sort. Which sensors to use, how many of them, and where to

place them are some of the design questions that are often difficult to answer, especially without knowing what these sensors are to be used for.

In this paper we review some recent results about *sensor minimization*. These results are interesting since observing an event can be costly in terms of time or energy: computation time must be spent to read and process the information provided by the sensor, and power is required to operate the sensor (as well as perform the computations). It is then essential that the sensors used really provide useful information. It is also important for the computer to discard any information given by a sensor that is not really needed.

In the case of a fixed set of observable events, it is not the case that all sensors *always* provide useful information and sometimes energy (used for sensor operation and computer treatment) is spent for nothing. For example, to detect a fault f in the system described by the automaton \mathcal{B} , Figure 1, page 3, an observer needs to watch only for event a initially, and watch for event b *only after a has occurred*. If the sequence $a.b$ occurs, for sure f has occurred and the observer can raise an alarm. If, on the other hand, event b is not observed after a , then f has not occurred. It is then not useful to switch on sensor b before observing event a .

B. Sensor Minimization and Fault Diagnosis

We focus our attention on sensor minimization, without looking at problems related to sensor placement, choosing between different types of sensors, and so on. We also focus on a particular observation problem, that of *fault diagnosis*. We believe, however, that the results we obtain are applicable to other contexts as well.

Fault diagnosis consists in observing a plant and detecting whether a fault has occurred or not. We follow the discrete-event system (DES) setting of [5] where the behavior of the plant is known and a model of it is available as a finite-state automaton over $\Sigma \cup \{\varepsilon, f\}$ where Σ is the set of potentially observable events, ε represents the unobservable events, and f is a special unobservable event that corresponds to the

*Preliminary versions of parts of this paper appeared in [1] and [2].

[†] Work supported by the French government under grant ANR-06-SETI.

faults¹. Checking *diagnosability* (whether a fault can be detected) for a given plant and a *fixed* set of observable events can be done in polynomial time [5], [6], [7]. In the general case, synthesizing a diagnoser involves determinization and thus cannot be done in polynomial time.

In this paper, we focus on *dynamic* observers. For results about sensor optimization with *static* observers, we refer the reader to [2].

In the dynamic observers' framework, we assume that an observer can decide after each new observation the set of events it is going to watch. We first prove that checking diagnosability with dynamic observers that are given by finite automata can be done in polynomial time. As a second aspect, we focus on the *dynamic observer synthesis problem*. We show that computing a *dynamic observer* for a given plant, can be reduced to a *game problem*. We further investigate optimization problems for dynamic observers and define a notion of *cost* of an observer. Finally we show how to compute an optimal (cost-wise) dynamic observer.

C. Related Work

To our knowledge, the problems of synthesizing dynamic observers for diagnosability, studied in Section III, have not been addressed previously in the literature. Consequently, the associated optimization problems, addressed in section IV, of computing an optimal observer is also original and new.

D. Organisation of the paper.

In Section II we fix notation and introduce finite automata with faults to model DES.

In Section III we introduce and study dynamic observers and show that the most permissive dynamic observer can be computed as the strategy in a safety 2-player game.

We also define a notion of cost for dynamic observers in Section IV and show that the cost of a given observer can be computed using Karp's algorithm. Finally, we define the optimal-cost observer synthesis problem and show it can be solved using Zwick and Paterson's result on graph games.

This paper contains no proofs and the interested reader may refer to [1], [2], [8] for the details.

II. PRELIMINARIES

A. Words and Languages

Let Σ be a finite alphabet and $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$. Σ^* is the set of finite words over Σ and contains ε which is also the empty word and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A *language* L is any subset of Σ^* . Given two words ρ, ρ' we denote $\rho.\rho'$ the concatenation of ρ and ρ' which is defined in the usual way. $|\rho|$ stands for the length of the word ρ (the length of the empty word is zero) and $|\rho|_\lambda$ with $\lambda \in \Sigma$ stands for the number of occurrences of λ in ρ . We also use the notation $|S|$ to denote the cardinality of a set S . Given $\Sigma_1 \subseteq \Sigma$, we define the *projection* operator on words, $\pi_{/\Sigma_1} : \Sigma^* \rightarrow \Sigma_1^*$,

¹Different types of faults could also be considered, by having different fault events f_1, f_2 , and so on. Our methods can be extended in a straightforward way to deal with multiple faults. We restrict our presentation to a single fault event for the sake of simplicity.

recursively as follows: $\pi_{/\Sigma_1}(\varepsilon) = \varepsilon$ and for $a \in \Sigma, \rho \in \Sigma^*$, $\pi_{/\Sigma_1}(a.\rho) = a.\pi_{/\Sigma_1}(\rho)$ if $a \in \Sigma_1$ and $\pi_{/\Sigma_1}(\rho)$ otherwise.

B. Finite Automata

Definition 1 (Finite Automaton) An automaton A is a tuple $(Q, q_0, \Sigma^\varepsilon, \delta)$ with Q a set of states², $q_0 \in Q$ is the initial state, $\delta \subseteq Q \times \Sigma^\varepsilon \times 2^Q$ is the transition relation. We write $q \xrightarrow{\lambda} q'$ if $q' \in \delta(q, \lambda)$. For $q \in Q$, $en(q)$ is the set of actions enabled at q .

If Q is finite, A is a finite automaton. An automaton is deterministic if for any $q \in Q$, $|\delta(q, \varepsilon)| = 0$ and for any $\lambda \neq \varepsilon$, $|\delta(q, \lambda)| \leq 1$. A labeled automaton A is a tuple $(Q, q_0, \Sigma, \delta, L)$ where (Q, q_0, Σ, δ) is an automaton and $L : Q \rightarrow P$ where P is a finite set of observations. ■

A run ρ from state s in A is a finite or infinite sequence of transitions

$$s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} s_2 \cdots s_{n-1} \xrightarrow{\lambda_n} s_n \cdots$$

s.t. $\lambda_i \in \Sigma^\varepsilon$ and $s_0 = s$. If ρ is finite and ends in s_n we let $tgt(\rho) = s_n$. The set of finite runs from s in A is denoted $Runs(s, A)$ and we define $Runs(A) = Runs(q_0, A)$. The *trace* of the run ρ , denoted $tr(\rho)$, is the word obtained by concatenating the symbols λ_i appearing in ρ , for those λ_i different from ε . A word w is *accepted* by A if $w = tr(\rho)$ for some $\rho \in Runs(A)$. The *language* $\mathcal{L}(A)$ of A is the set of words accepted by A .

Let $f \notin \Sigma^\varepsilon$ be a fresh letter that corresponds to the fault action, $\Sigma^{\varepsilon, f} = \Sigma^\varepsilon \cup \{f\}$ and $A = (Q, q_0, \Sigma^{\varepsilon, f}, \delta)$. Given $R \subseteq Runs(A)$, $Tr(R) = \{tr(\rho) \text{ for } \rho \in R\}$ is the set of traces of the runs in R . A run ρ is *k-faulty* if there is some $1 \leq i \leq n$ s.t. $\lambda_i = f$ and $n - i \geq k$. Notice that ρ can be either finite or infinite: if it is infinite, $n = \infty$ and $n - i \geq k$ always holds. $Faulty_{\geq k}(A)$ is the set of *k-faulty* runs of A . A run is *faulty* if it is *k-faulty* for some $k \in \mathbb{N}$ and $Faulty(A)$ denotes the set of faulty runs. It follows that $Faulty_{\geq k+1}(A) \subseteq Faulty_{\geq k}(A) \subseteq \cdots \subseteq Faulty_{\geq 0}(A) = Faulty(A)$. Finally, $NonFaulty(A) = Runs(A) \setminus Faulty(A)$ is the set on *non-faulty* runs of A . We let $Faulty_{\geq k}^{tr}(A) = Tr(Faulty_{\geq k}(A))$ and $NonFaulty^{tr}(A) = Tr(NonFaulty(A))$ be the sets of traces of faulty and non-faulty runs.

We assume that each faulty run of A of length n can be extended into a run of length $n + 1$. This is required for technical reasons (in order to guarantee that the set of faulty runs where sufficient time has elapsed after the fault is well-defined) and can be achieved by adding ε loop-transitions to each deadlock state of A . Notice that this transformation does not change the observations produced by the plant, thus, any observer synthesized for the transformed plant also applies to the original one.

C. Product of Automata

The product of automata with ε -transitions is defined in the usual way: the automata synchronize on common labels except for ε . Let $A_1 = (Q_1, q_0^1, \Sigma_1^\varepsilon, \rightarrow_1)$ and $A_2 = (Q_2, q_0^2, \Sigma_2^\varepsilon,$

²In this paper we often use finite automata that generate prefix-closed languages, hence we do not need to use a set of final or accepting states.

\rightarrow_2). The product of A_1 and A_2 is the automaton $A_1 \times A_2 = (Q, q_0, \Sigma, \rightarrow)$ where:

- $Q = Q_1 \times Q_2$,
- $q_0 = (q_0^1, q_0^2)$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is defined by $(q_1, q_2) \xrightarrow{\sigma} (q'_1, q'_2)$ if:
 - either $\sigma \in \Sigma_1 \cap \Sigma_2$ and $q_k \xrightarrow{\sigma} q'_k$, for $k = 1, 2$,
 - or $\sigma \in (\Sigma_i \setminus \Sigma_{3-i}) \cup \{\varepsilon\}$ and $q_i \xrightarrow{\sigma} q'_i$ and $q'_{3-i} = q_{3-i}$, for $i = 1$ or $i = 2$.

III. FAULT DIAGNOSIS WITH DYNAMIC OBSERVERS

In this section we introduce *dynamic observers*. They can choose after each new observation the set of events they are going to watch for. To illustrate why dynamic observers can be useful consider the following example.

Example 1 (Dynamic Observation) Assume we want to detect faults in automaton \mathcal{B} of Figure 1. A static diagnoser that observes $\Sigma = \{a, b\}$ can detect faults. However, no proper subset of Σ can be used to detect faults in \mathcal{B} . Thus the minimum cardinality of the set of observable events for diagnosing \mathcal{B} is 2 i.e., a static observer will have to monitor two events during the execution of the DES. This means that an observer will have to be receptive to at least two inputs at each point in time to detect a fault in \mathcal{B} . One can think of being receptive as switching on a device to sense an event. This consumes energy. We can be more efficient using a dynamic observer, that only turns on sensors when needed, thus saving energy. In the case of \mathcal{B} , this can be done as follows: in the beginning we only switch on the a -sensor; once an a occurs the a -sensor is switched off and the b -sensor is switched on. Compared to the previous diagnosers we use half as much energy.

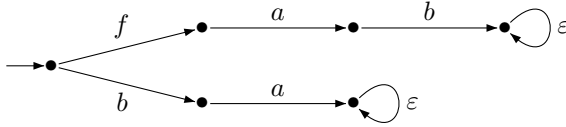


Fig. 1. The automaton \mathcal{B}

A. Dynamic Observers

We formalize the above notion of dynamic observation using *observers*. The choice of the events to observe can depend on the choices the observer has made before and on the observations it has made. Moreover an observer may have *unbounded* memory.

Definition 2 (Observer) An observer Obs over Σ is a deterministic labeled automaton $\text{Obs} = (S, s_0, \Sigma, \delta, L)$, where S is a (possibly infinite) set of states, $s_0 \in S$ is the initial state, Σ is the set of observable events, $\delta : S \times \Sigma \rightarrow S$ is the transition function (a total function), and $L : S \rightarrow 2^\Sigma$ is a labeling function that specifies the set of events that the observer wishes to observe when it is at state s . We require for any state s and any $a \in \Sigma$, if $a \notin L(s)$ then $\delta(s, a) = s$:

this means the observer does not change its state when an event it has chosen not to observe occurs. ■

As an observer is deterministic we use the notation $\delta(s_0, w)$ to denote the state s reached after reading the word w and $L(\delta(s_0, w))$ is the set of events Obs observes after w .

An observer implicitly defines a *transducer* that consumes an input event $a \in \Sigma$ and, depending on the current state s , either outputs a (when $a \in L(s)$) and moves to a new state $\delta(s, a)$, or outputs ε , (when $a \notin L(s)$) and remains in the same state waiting for a new event. Thus, an observer defines a mapping Obs from Σ^* to Σ^* (we use the same name “Obs” for the automaton and the mapping). Given a run ρ , $\text{Obs}(\pi_{/\Sigma}(\text{tr}(\rho)))$ is the output of the transducer on ρ . It is called the *observation* of ρ by Obs . We next provide an example of a particular case of observer which can be represented by a finite-state machine.

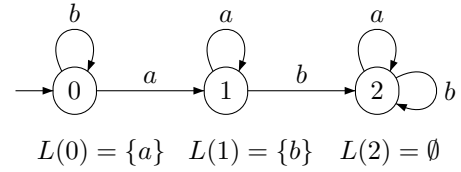


Fig. 2. A finite-state observer Obs

Example 2 Let Obs be the observer of Figure 2. Obs maps the following inputs as follows: $\text{Obs}(\text{baab}) = ab$, $\text{Obs}(\text{bababbaab}) = ab$, $\text{Obs}(\text{bbbbba}) = a$ and $\text{Obs}(\text{bbaaaa}) = a$. If Obs operates on the DES \mathcal{B} of Figure 1 and \mathcal{B} generates $f.a.b$, Obs will have as input $\pi_{/\Sigma}(f.a.b) = a.b$ with $\Sigma = \{a, b\}$. Consequently the observation of Obs is $\text{Obs}(\pi_{/\Sigma}(f.a.b)) = a.b$.

B. Fault Diagnosis with Dynamic Diagnosers

Definition 3 ((Obs, k)-diagnoser) Let A be a finite automaton over $\Sigma^{\varepsilon, f}$ and Obs be an observer over Σ . $D : \Sigma^* \rightarrow \{0, 1\}$ is an (Obs, k) -diagnoser for A if

- $\forall \rho \in \text{NonFaulty}(A), D(\text{Obs}(\pi_{/\Sigma}(\text{tr}(\rho)))) = 0$ and
- $\forall \rho \in \text{Faulty}_{\geq k}(A), D(\text{Obs}(\pi_{/\Sigma}(\text{tr}(\rho)))) = 1$. ■

A is (Obs, k) -diagnosable if there is an (Obs, k) -diagnoser for A . A is Obs -diagnosable if there is some k such that A is (Obs, k) -diagnosable.

If a diagnoser always selects Σ as the set of observable events, it is a static observer and (Obs, k) -diagnosability amounts to the standard (Σ, k) -diagnosis problem [5].

As for Σ -diagnosability, we have the following equivalence for dynamic observers: A is (Obs, k) -diagnosable iff

$$\text{Obs}(\pi_{/\Sigma}(\text{Faulty}_{\geq k}^{\text{tr}}(A))) \cap \text{Obs}(\pi_{/\Sigma}(\text{NonFaulty}^{\text{tr}}(A))) = \emptyset.$$

Problem 1 (Finite-State Obs-Diagnosability)

INPUT: A , Obs a finite-state observer.

PROBLEM:

- Is A Obs -diagnosable?
- If the answer to (A) is “yes”, compute the minimum k such that A is (Obs, k) -diagnosable.

Theorem 1 *Problem 1 is in P.*

To prove Theorem 1 we build a *product automaton*³ $A \otimes \text{Obs}$ such that: A is (Obs, k) -diagnosable $\iff A \otimes \text{Obs}$ is (Σ, k) -diagnosable. Given two finite automata $A = (Q, q_0, \Sigma^{\varepsilon, f}, \rightarrow)$ and $\text{Obs} = (S, s_0, \Sigma, \delta, L)$, the automaton $A \otimes \text{Obs} = (Q \times S, (q_0, s_0), \Sigma^{\varepsilon, f}, \rightarrow)$ is defined as follows:

- $(q, s) \xrightarrow{\beta} (q', s')$ iff $\exists \lambda \in \Sigma$ s.t. $q \xrightarrow{\lambda} q'$, $s' = \delta(s, \lambda)$ and $\beta = \lambda$ if $\lambda \in L(s)$, $\beta = \varepsilon$ otherwise;
- $(q, s) \xrightarrow{\lambda} (q', s)$ iff $\exists \lambda \in \{\varepsilon, f\}$ s.t. $q \xrightarrow{\lambda} q'$.

The number of states of $A \otimes \text{Obs}$ is at most $|Q| \times |S|$ and the number of transitions is bounded by the number of transitions of A . Hence the size of the product is polynomial in the size of the input $|A| + |\text{Obs}|$. Checking that $A \otimes \text{Obs}$ is diagnosable can be done in polynomial time and Problem 1.(A) is in P.

Example 3 Let \mathcal{B} be the DES given in Figure 1 and Obs the observer of Figure 2. The product $A \otimes \text{Obs}$ used in the above proof is given in Figure 3.

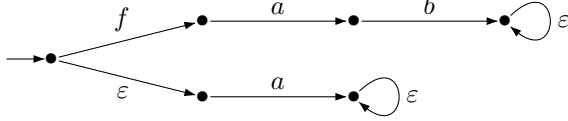


Fig. 3. The product $A \otimes \text{Obs}$

For Problem 1, we have assumed that an observer was given. It would be even better if we could *synthesize* an observer Obs such that the plant is Obs -diagnosable. Before attempting to synthesize such an observer, we should first check that the plant is Σ -diagnosable: if it is not, then obviously no such observer exists; if the plant is Σ -diagnosable, then the trivial observer that observes all events in Σ at all times works⁴. As a first step towards synthesizing non-trivial observers, we can attempt to compute the set of *all* valid observers, which includes the trivial one but also non-trivial ones (if they exist).

Problem 2 (Dynamic-Diagnosability)

INPUT: A .

PROBLEM: Compute the set \mathcal{O} of all observers such that A is Obs -diagnosable iff $\text{Obs} \in \mathcal{O}$.

We do not have a solution to the above general problem. Instead, we introduce a restricted variant:

Problem 3 (Dynamic- k -Diagnosability)

INPUT: A , $k \in \mathbb{N}$.

PROBLEM: Compute the set \mathcal{O} of all observers such that A is (Obs, k) -diagnosable iff $\text{Obs} \in \mathcal{O}$.

³ We use \otimes to clearly distinguish this product from the usual synchronous product \times .

⁴ Notice that this also shows that existence of an observer implies existence of a finite-state observer, since the trivial observer is finite-state.

C. Problem 3 as a Game Problem

To solve Problem 3 we reduce it to a *safety* 2-player game. In short, the reduction we propose is the following:

- Player 1 chooses the set of events it wishes to observe, then it hands over to Player 2;
- Player 2 chooses an event and tries to produce a run which is the observation of a k -faulty run and a non-faulty run.

Player 2 wins if he can produce such a run. Otherwise Player 1 wins. Player 2 has complete information of Player 1's moves (i.e., it can observe the sets that Player 1 chooses to observe). Player 1, on the other hand, only has partial information of Player 2's moves because not all events are observable (details follow). Let $A = (Q, q_0, \Sigma^{\varepsilon, f}, \rightarrow)$ be a finite automaton. To define the game, we use two copies of automaton A : A_1^k and A_2 . The accepting states of A_1^k are those corresponding to runs of A which are faulty and where more than k steps occurred after the fault. A_2 is a copy of A where the f -transitions have been removed. The game we are going to play is the following (see Figure 4, Player 1 states are depicted with square boxes and Player 2 states with round shapes):

- 1) the game starts in an state (q_1, q_2) corresponding to the initial state of the product of A_1^k and A_2 . Initially, it is Player 1's turn to play. Player 1 chooses a set of events he is going to observe i.e., a subset X of Σ and hands it over to Player 2;
- 2) assume the automata A_1^k and A_2 are in states (q_1, q_2) . Player 2 can change the state of A_1^k and A_2 by:
 - a) firing an action (like $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ in Figure 4) which is not in X in either A_1^k or A_2 (no synchronization). In this case a new state (q, q') is reached and Player 2 can play again from this state;
 - b) firing an action in X (like σ_1, σ_2 in Figure 4): to do this both A_1^k and A_2 must be in a state where λ is possible (synchronization); after the action is fired a new state (q'_1, q'_2) is reached: now it is Player 1's turn to play, and the game continues as in step 1 above from the new state (q'_1, q'_2) .

Player 2 wins if he can reach a state (q_1, q_2) in $A_1^k \times A_2$ where q_1 is an accepting state of A_1^k (this means that Player 1 wins if it can avoid ad infinitum this set of states). In this sense this is a safety game for Player 1 (and a reachability game for Player 2). Formally, the game $G_A = (S_1 \uplus S_2, s_0, \Sigma_1 \uplus \Sigma_2, \delta)$ is defined as follows (\uplus denotes union of disjoint sets):

- $S_1 = (Q \times \{-1, \dots, k\}) \times Q$ is the set of Player 1 states; a state $((q_1, j), q_2) \in S_1$ indicates that A_1^k is in state q_1 , j steps have occurred after a fault, and q_2 is the current state of A_2 . If no fault has occurred, $j = -1$ and if more than k steps occurred after the fault, we use $j = k$.
- $S_2 = (Q \times \{-1, \dots, k\}) \times Q \times 2^\Sigma$ is the set of Player 2 states. For a state $((q_1, j), q_2, X) \in S_2$, the triple $((q_1, j), q_2)$ has the same meaning as for S_1 , and X is the set of moves Player 1 has chosen to observe on its last move.

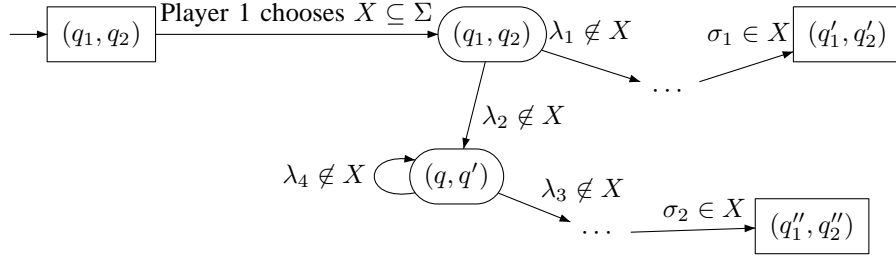


Fig. 4. Game reduction for problem 3

- $s_0 = ((q_0, -1), q_0)$ is the initial state of the game belonging to Player 1;
- $\Sigma_1 = 2^\Sigma$ is the set of moves of Player 1; $\Sigma_2 = \Sigma^\varepsilon$ is the set of moves of Player 2 (as we encode the fault into the state, we do not need to distinguish f from ε).
- the transition relation $\delta \subseteq (S_1 \times \Sigma_1 \times S_2) \cup (S_2 \times \{\varepsilon\} \times S_2) \cup (S_2 \times \Sigma \times S_1)$ is defined by:
 - Player 1 moves: let $\sigma \in \Sigma_1$ and $s_1 \in S_1$. Then $(s_1, \sigma, (s_1, \sigma)) \in \delta$.
 - Player 2 moves: a move of Player 2 is either a silent move (ε) i.e., a move of A_1^k or A_2 or a joint move of A_1^k and A_2 with an observable action in X . Consequently, a *silent* move $((q_1, i), q_2, X), \varepsilon, (q_1', j), q_2', X)$ is in δ if one of the following conditions holds:
 - 1) either $q_2' = q_2, q_1 \xrightarrow{\ell} q_1'$ is a step of $A_1^k, \ell \notin X$, and if $i \geq 0$ then $j = \min(i+1, k)$; if $i = -1$ and $\ell = f$ $j = 0$ otherwise $j = i$.
 - 2) either $q_1' = q_1, q_2 \xrightarrow{\ell} q_2'$ is a step of $A_2, \ell \notin X$ (and $\ell \neq f$), and if $i \geq 0$ then $j = \min(i+1, k)$, otherwise $j = i$.

A *visible* move can be taken by Player 2 if both A_1^k and A_2 agree on doing such a move. In this case the game proceeds to a Player 1 state: $((q_1, i), q_2, X), \ell, ((q_1', j), q_2') \in \delta$ if $\ell \in X, q_1 \xrightarrow{\ell} q_1'$ is a step of $A_1^k, q_2 \xrightarrow{\ell} q_2'$ is a step of A_2 , and if $i \geq 0$ then $j = \min(i+1, k)$, otherwise $j = i$.

We can show that for any observer O s.t. A is (O, k) -diagnosable, there is a strategy $f(O)$ for Player 1 in G_A s.t. $f(O)$ is *trace-based* and winning. A *strategy* for Player 1 is a mapping $f : \text{Runs}(G_A) \rightarrow \Sigma_1$ that associates a move $f(\rho)$ in Σ_1 to each run ρ in G_A that ends in an S_1 -state. A strategy f is *trace-based* if given two runs ρ, ρ' , if $\text{tr}(\rho) = \text{tr}(\rho')$ then $f(\rho) = f(\rho')$. Conversely, for any trace-based winning strategy f (for Player 1), we can build an observer $O(f)$ s.t. A is $(O(f), k)$ -diagnosable.

Let $O = (S, s_0, \Sigma, \delta, L)$ be an observer for A . We define the strategy $f(O)$ on finite runs of G_A ending in a Player 1 state by: $f(O)(\rho) = L(\delta(s_0, \pi_{/\Sigma}(\text{tr}(\rho))))$. The intuition is that we take the run ρ in G_A , take the trace of ρ (choices of Player 1 and moves of Player 2) and remove the choices of Player 1. This gives a word in Σ^* . The strategy for Player 1 for ρ is the set of events the observer O chooses to observe

after reading $\pi_{/\Sigma}(\text{tr}(\rho))$ i.e., $L(\delta(s_0, \pi_{/\Sigma}(\text{tr}(\rho))))$.

Conversely, with each trace-based strategy f of the game G_A we can associate an automaton $O(f) = (S, s_0, \Sigma, \delta, L)$ defined by:

- $S = \{\pi_{/\Sigma}(\text{tr}(\rho)) \mid \rho \in \text{Out}(G_A, f) \text{ and } \text{tgt}(\rho) \in S_1\}$;
- $s_0 = \varepsilon$;
- $\delta(v, \ell) = v'$ if $v \in S, v' = v.\ell$ and there is a run $\rho \in \text{Out}(G_A, f)$ with $\rho = q_0 \xrightarrow{X_0} q_0^1 \xrightarrow{\varepsilon^*} q_0^{n_0} \xrightarrow{\lambda_1} q_1 \xrightarrow{X_1} q_1^1 \xrightarrow{\varepsilon^*} q_1^{n_1} \xrightarrow{\lambda_2} q_2 \cdots q_{k_1} \xrightarrow{\varepsilon^*} q_{k-1}^{n_{k-1}} \xrightarrow{\lambda_k} q_k$ with each $q_i \in S_1, q_i^j \in S_2, v = \pi_{/\Sigma}(\text{tr}(\rho))$, and $\rho \xrightarrow{X_k} q_k^1 \xrightarrow{\varepsilon^*} q_k^{n_k} \xrightarrow{\ell} q_{k+1}$ with $q_{k+1} \in S_1, \ell \in X_k$.
 $\delta(v, \ell) = v$ if $v \in S$ and $\ell \notin f(\rho)$;
- $L(v) = f(\rho)$ if $v = \pi_{/\Sigma}(\text{tr}(\rho))$.

Using the previous definitions and constructions we obtain the following theorems:

Theorem 2 Let O be an observer s.t. A is (O, k) -diagnosable. Then $f(O)$ is a trace-based winning strategy in G_A .

Theorem 3 Let f be a trace-based winning strategy in G_A . Then $O(f)$ is an observer and A is $(O(f), k)$ -diagnosable.

The result on a game like G_A is that, if there is a winning trace-based strategy for Player 1, then there is a most permissive strategy \mathcal{F}_A which has finite memory. It can be represented by a finite automaton $S_{\mathcal{F}_A} = (W_1 \uplus W_2, s_0, \Sigma \cup 2^\Sigma, \Delta_A)$ s.t. $\Delta_A \subseteq (W_1 \times 2^\Sigma \times W_2) \cup (W_2 \times \Sigma \times W_1)$ which has size exponential in the size of G_A . For a given run $\rho \in (\Sigma \cup 2^\Sigma)^*$ ending in a W_1 -state, we have $\mathcal{F}_A(w) = \text{en}(\Delta_A(s_0, w))$.

D. Most Permissive Observer

We now define the notion of a most *permissive* observer and show the existence of a most permissive observer for a system in case A is diagnosable. \mathcal{F}_A is the mapping defined at the end of the previous section.

For an observer $O = (S, s_0, \Sigma, \delta, L)$ and $w \in \Sigma^*$ we let $L(w)$ be the set $L(\delta(s_0, w))$: this is the set of events O chooses to observe on input w . Given a word $\rho \in \pi_{/\Sigma}(\mathcal{L}(A))$, we recall that $O(\rho)$ is the observation of ρ by O . Assume $O(\rho) = a_0 \cdots a_k$. Let $\bar{\rho} = L(\varepsilon).\varepsilon.L(a_0).a_0 \cdots L(O(\rho)(k)).a_k$ i.e., $\bar{\rho}$ contains the history of what O has chosen to observe at each step and the events that occurred after each choice.

Let $\mathcal{O} : (2^\Sigma \times \Sigma^\varepsilon)^+ \rightarrow 2^{2^\Sigma}$. By definition \mathcal{O} is the most permissive observer for (A, k) if the following holds:

$$\begin{aligned} O = (S, s_0, \Sigma, \delta, L) \\ \text{is an observer and} \\ \text{and } A \text{ is } (O, k)\text{-diagnosable} \end{aligned} \iff \forall w \in \Sigma^*, L(\delta(s_0, w)) \in \mathcal{O}(\overline{w})$$

The definition of the most permissive observer states that:

- any good observer O (one such that A is (O, k) -diagnosable) must choose a set of observable events in $\mathcal{O}(\overline{w})$ on input w ;
- if an observer chooses its set of observable events in $\mathcal{O}(\overline{w})$ on input w , then it is a good observer.

Assume A is (Σ, k) -diagnosable. Then there is an observer O s.t. A is (O, k) -diagnosable because the constant observer that observes Σ is a solution. By Theorem 2, there is a trace-based winning strategy for Player 1 in G_A .

Theorem 4 \mathcal{F}_A is the most permissive observer.

This enables us to solve Problem 3 and compute a finite representation of the set \mathcal{O} of all observers such that A is (O, k) -diagnosable iff $O \in \mathcal{O}$. Computing \mathcal{F}_A can be done in $O(2^{|G_A|})$. The size of G_A is quadratic in $|A|$, linear in the size of k , and exponential in the size of Σ i.e., $|G_A| = O(|A|^2 \times 2^{|\Sigma|} \times |k|)$. This means that computing \mathcal{F}_A can be done in exponential time in the size of A and k and doubly exponential time in the size of Σ .

The computation of a generic diagnoser associated with the most permissive observer can be done as well. This diagnoser is the *most permissive dynamic* diagnoser and contains all the choices a dynamic diagnoser can make to be able to diagnose a plant.

IV. OPTIMAL DYNAMIC OBSERVERS

In this section we define a notion of cost for observers. This will allow us to compare observers w.r.t. to this criterion and later on to synthesize an optimal observer. The notion of cost we are going to use is inspired by *weighted automata*.

A. Weighted Automata & Karp's Algorithm

The notion of cost for automata has already been defined and algorithms to compute some optimal values related to this model are described in many papers. We recall here the results of [9] which will be used later.

Definition 4 (Weighted Automaton) A weighted automaton is a pair (A, w) s.t. $A = (Q, q_0, \Sigma, \delta)$ is a finite automaton and $w : Q \rightarrow \mathbb{N}$ associates a weight with each state. ■

Definition 5 (Mean Cost) Let $\rho = q_0 \xrightarrow{a_2} q_1 \xrightarrow{a_1} \dots \xrightarrow{a_n} q_n$ be a run of A . The mean cost of ρ is

$$\mu(\rho) = \frac{1}{n+1} \times \sum_{i=0}^n w(q_i).$$

We remind that the length of $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$ is $|\rho| = n$. We assume that A is complete w.r.t. Σ (and $\Sigma \neq \emptyset$) and thus contains at least one run for any arbitrary length n . Let $Runs^n(A)$ be the set of runs of length n in $Runs(A)$. The *maximum mean-weight* of the runs of length n for A is $\nu(A, n) = \max\{\mu(\rho) \text{ for } \rho \in Runs^n(A)\}$. The *maximum mean weight* of A is $\nu(A) = \limsup_{n \rightarrow \infty} \nu(A, n)$. Actually the value $\nu(A)$ can be computed using Karp's maximum mean-weight cycle algorithm [9] on weighted graphs. If $c = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$ is a cycle of A i.e., $s_0 = s_n$, the *mean weight* of the cycle c is $\mu(c) = \frac{1}{n+1} \cdot \sum_{i=0}^n w(s_i)$. The *maximum mean-weight cycle* of A is the value $\nu^*(A) = \max\{\mu(c) \text{ for } c \text{ a cycle of } A\}$. As stated in [10], for weighted automata, the mean-weight cycle value is the value that determines the mean-weight value: $\nu(A) = \limsup_{n \rightarrow \infty} \nu(A, n) = \lim_{n \rightarrow \infty} \nu(A, n) = \nu^*(A)$.

B. Cost of a Dynamic Observer

Let $\text{Obs} = (S, s_0, \Sigma, \delta, L)$ be an observer and $A = (Q, q_0, \Sigma^{\varepsilon, f}, \rightarrow)$. We would like to define a notion of *cost* for observers in order to select an optimal one among all of those which are valid, i.e., s.t. A is (Obs, k) -diagnosable. Intuitively this notion of cost should imply that the more events we observe at each time, the more expensive it is.

There is not one way of defining a notion of cost and the reader is referred to [1] for a discussion on this subject.

The cost of a word w is given by:

$$\text{Cost}(w) = \frac{\sum_{i=0}^{|w|-1} |L(\delta(s_0, w(i)))|}{|w| + 1}$$

with $n = |w|$.

We now show how to define and compute the cost of an observer Obs that observes a DES A .

Given a run $\rho \in Runs(A)$, the observer only processes $\pi_{/\Sigma}(tr(\rho))$ (ε and f -transitions are not processed). To have a consistent notion of costs that takes into account the logical time elapsed from the beginning, we need to take into account one way or another the number of *steps* of ρ (the length of ρ) even if some of them are non observable. A simple way to do this is to consider that ε and f are now observable events, let's say u , but that the observer never chooses to observe them. Indeed we assume we have already checked that A is (Obs, k) -diagnosable, and the problem is now to compute the cost of the observer we have used.

Definition 6 (Cost of a Run) Given a run $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n \in Runs(A)$, let $w_i = \text{Obs}(\pi_{/\Sigma}(tr(\rho(i))))$, $0 \leq i \leq n$. The cost of $\rho \in Runs(A)$ is defined by:

$$\text{Cost}(\rho, A, \text{Obs}) = \frac{1}{n+1} \cdot \sum_{i=0}^n |L(\delta(s_0, w_i))|.$$

We recall that $Runs^n(A)$ is the set of runs of length n in $Runs(A)$. The cost of the runs of length n of A is

$$\text{Cost}(n, A, \text{Obs}) = \max\{\text{Cost}(\rho, A, \text{Obs}) \text{ for } \rho \in Runs^n(A)\}.$$

The cost of the pair (Obs, A) is

$$\text{Cost}(A, \text{Obs}) = \limsup_{n \rightarrow \infty} \text{Cost}(n, A, \rho).$$

Notice that $\text{Cost}(n, A, \text{Obs})$ is defined for each n because we have assumed A generates runs of arbitrary large length.

As emphasised previously, in order to compute $\text{Cost}(n, A, \text{Obs})$ we consider that ε and f are now observable events, say u , but that the observer never chooses to observe them. Let $\text{Obs}^+ = (S, s_0, \Sigma^u, \delta', L)$ where δ' is δ augmented with u -transitions that loop on each state $s \in S$. Let A^+ be A where ε and f transitions are renamed u . Let $A^+ \times \text{Obs}^+$ be the synchronized product of A^+ and Obs^+ . $A^+ \times \text{Obs}^+ = (Z, z_0, \Sigma^u, \Delta)$ is complete w.r.t. Σ^u and we let $w(q, s) = |L(s)|$ so that $(A^+ \times \text{Obs}^+, w)$ is a weighted automaton.

Theorem 5 $\text{Cost}(A, \text{Obs}) = \nu^*(A^+ \times \text{Obs}^+)$.

Thus we can compute the cost of a given pair (A, Obs) : this can be done using Karp's maximum mean weight cycle algorithm [9] on weighted graphs. This algorithm is polynomial in the size of the weighted graph and thus:

Theorem 6 *Computing the cost of (A, Obs) is in P .*

Remark 1 *Notice that instead of the values $|L(s)|$ we could use any mapping from states of Obs to \mathbb{Z} and consider that the cost of observing $\{a, b\}$ is less than observing a .*

C. Optimal Dynamic Diagnoser

In this section, we focus on the problem of computing a best observer in the sense that diagnosing the DES with it has minimal cost. We address the following problem:

Problem 4 (Bounded Cost Observer)

INPUT: $A, k \in \mathbb{N}$ and $c \in \mathbb{N}$.

PROBLEM:

- (A). *Is there an observer Obs s.t. A is (Obs, k) -diagnosable and $\text{Cost}(\text{Obs}) \leq c$?*
- (B). *If the answer to (A) is "yes", compute a witness optimal observer Obs with $\text{Cost}(\text{Obs}) \leq c$.*

Theorem 4, page 6 establishes that there is a most permissive observer \mathcal{F}_A in case A is (Σ, k) -diagnosable and it can be computed in exponential time in the size of A and k , doubly exponential time in $|\Sigma|$, and has size exponential in A and k , and doubly exponential in $|\Sigma|$. Moreover the most permissive observer \mathcal{F}_A can be represented by a finite state machine $S_{\mathcal{F}_A} = (\{0, 2 \cdots, l\} \cup (\{1, 3, \cdots, 2l' + 1\} \times 2^\Sigma), 0, \Sigma \cup 2^\Sigma, \delta)$ which has the following properties:

- even states are states where the observer chooses a set of events to observe;
- odd states $(2i + 1, X)$ are states where the observer waits for an observable event in X to occur;
- if $\delta(2i, X) = (2i' + 1, X)$ with $X \in 2^\Sigma$, it means that from an even state $2i$, the automaton $S_{\mathcal{F}_A}$ can select a set X of events to observe. The successor state is an odd

state together with the set X of events that are being observed;

- if $\delta((2i + 1, X), a) = 2i'$ with $a \in X$, it means that from $(2i + 1, X)$, $S_{\mathcal{F}_A}$ is waiting for an observable event to occur. When some occurs it switches to an even state.

By definition of \mathcal{F}_A , any observer O s.t. A is (O, k) -diagnosable must select a set of observable events in $\mathcal{F}_A(\text{tr}(\overline{w}))$ after having observed $w \in \pi_{/\Sigma}(\mathcal{L}(A))$.

To compute an optimal observer, we use a result by Zwicky and Paterson [10] on *weighted graph games*.

Definition 7 (Weighted Graph) *A weighted directed graph is a pair (G, w) s.t. $G = (V, E)$ is a directed graph and $w : E \rightarrow \{-W, \dots, 0, \dots, W\}$ assigns an integral weight to each edge of G with $W \in \mathbb{N}$. We assume that each vertex $v \in V$ is reachable from a unique source vertex v_0 and has at least one outgoing transition.* ■

Definition 8 (Weighted Graph Game) *A weighted graph game $G = (V, E)$ is a bipartite weighted graph with $V = V_1 \cup V_2$ and $E = E_1 \cup E_2$, $E_1 \subseteq V_1 \times V_2$ and $E_2 \subseteq V_2 \times V_1$. We assume the initial vertex v_0 of G belongs to V_1 .* ■

Vertices V_i are Player i 's vertex. A weighted graph game is a turn based game in which the turn alternates between Player 1 and Player 2. The game starts at a vertex $v_0 \in V_1$. Player 1 chooses an edge $e_1 = (v_0, v_1)$ and then Player 2 chooses an edge $e_2 = (v_1, v_2)$ and so on and they build an infinite sequence of edges. Player 1 wants to maximise $\liminf_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=1}^n w(e_i)$ and Player 2 wants to minimize $\limsup_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=1}^n w(e_i)$.

One of the result of [10] is that there is a rational value $\nu \in \mathbb{Q}$ s.t. Player 1 has a strategy to ensure $\liminf_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=1}^n w(e_i) \geq \nu$ and Player 2 has a strategy to ensure that $\limsup_{n \rightarrow \infty} \frac{1}{n} \cdot \sum_{i=1}^n w(e_i) \leq \nu$. ν is called the value of the game.

In summary the results by Zwicky and Paterson [10] we are going to use are:

- there is a value $\nu \in \mathbb{Q}$, called the *value of the game* s.t. Player 1 has a strategy to ensure that $\liminf_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n w(e_i) \geq \nu$ and Player 2 has a strategy to ensure that $\limsup_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n w(e_i) \leq \nu$; this value can be computed in $O(|V|^3 \times |E| \times W)$ where W is the range of the weight function (assuming the weights are in the interval $[-W..W]$). Note that deciding whether this value satisfies $\nu \bowtie c$ for $\bowtie \in \{=, <, >\}$ for $c \in \mathbb{Q}$ can be done in $O(|V|^2 \times |E| \times W)$.
- there are optimal memoryless strategies for both players that can be computed in $O(|V|^4 \times |E| \times \log(|E|/|V|) \times W)$.

To solve Problem 4, we use the most permissive observer \mathcal{F}_A we computed in section III-D. Given A and \mathcal{F}_A , we build a weighted graph game $G(A, \mathcal{F}_A)$ s.t. the value of the game is the optimal cost for the set of all observers. Moreover an optimal observer can be obtained by taking an optimal memoryless strategy in $G(A, \mathcal{F}_A)$.

To build $G(A, \mathcal{F}_A)$ we use the same idea as in section IV-B: we replace ε and f transitions in A by u obtaining A^+ . We also modify \mathcal{F}_A to obtain a weighted graph game (\mathcal{F}_A^+, w) by adding transitions so that each state $2k+1$ is complete w.r.t. Σ^u . This is done as follows:

- from each $(2i+1, X)$ state, create a new even state i.e., pick some $2i'$ that has not already been used. Add transitions $((2i+1, X), \sigma, 2i')$ for each $\sigma \in \Sigma^u \setminus \text{en}(2i+1, X)$. Add also a transition $(2i', X, (2i+1, X))$. This step means that if a A produces an event and it is not observable, \mathcal{F}_A^+ just reads the event and makes the same choice again.
- the weight of a transition $(2i, X, (2i'+1, X))$ is $|X|$.

The game $G(A, \mathcal{F}_A)$ is then $A^+ \times \mathcal{F}_A^+$. This way we can obtain a weighted graph game $WG(A, \mathcal{F}_A)$ by abstracting away the labels of the transitions. Notice that it still enables us to convert any strategy in $WG(A, \mathcal{F}_A)$ to a strategy in \mathcal{F}_A . A strategy in $WG(A, \mathcal{F}_A)$ will define an edge $(2i, (2i'+1, X))$ to take. As the target vertex contains the set of events we chose to observe we can define a corresponding strategy in \mathcal{F}_A .

By construction of $G(A, \mathcal{F}_A)$ and the definition of the value of a weighted graph game, the value of the game is the optimal cost for the set of all observers O s.t. A is (O, k) -diagnosable.

Assume A has n states and m transitions. From Theorem 4 we know that \mathcal{F}_A has at most $O(2^{n^2} \times 2^k \times 2^{2^{|\Sigma|}})$ states and $O(2^{n^2} \times 2^k \times 2^{2^{|\Sigma|}} \times n^2 \times k \times m)$ transitions. Hence $G(A, \mathcal{F}_A)$ has at most $O(n \times 2^{n^2} \times 2^k \times 2^{2^{|\Sigma|}})$ vertices and $O(m \times 2^{n^2} \times 2^k \times 2^{2^{|\Sigma|}})$ edges. To make the game complete we may add at most half the number of states and hence $WG(A, \mathcal{F}_A)$ has the same size. We thus obtain the following results:

Theorem 7 *Problem 4 can be solved in time $O(|\Sigma| \times m \times 2^{n^2} \times 2^k \times 2^{2^{|\Sigma|}})$.*

We can even solve the optimal cost computation problem:

Problem 5 (Optimal Cost Observer)

INPUT: $A, k \in \mathbb{N}$.

PROBLEM: Compute the least value m s.t. there exists an observer Obs s.t. A is (Obs, k) -diagnosable and $\text{Cost}(\text{Obs}) \leq m$.

Theorem 8 *Problem 5 can be solved in time $O(|\Sigma| \times m \times 2^{n^2} \times 2^k \times 2^{2^{|\Sigma|}})$.*

A consequence of Theorem 8 and Zwick and Paterson's results is that the cost of the optimal observer is a rational number.

V. CONCLUSIONS

In this paper we have addressed sensor minimization problems in the context of fault diagnosis, using dynamic observers. We proved that, for an observer given by a finite automaton, diagnosability can be checked in polynomial time

(as in the case of static observers). We also solved a synthesis problem of dynamic observers and showed that a most-permissive dynamic observer can be computed in doubly-exponential time, provided an upper bound on the delay needed to detect a fault is given. Finally we have defined a notion of cost for dynamic observers and shown how to compute the minimal-cost observer that can be used to detect faults within a given delay.

There are several directions we are currently investigating.

Problem 2 has not been solved so far. The major impediment to solve it is that the reduction we propose in section III yields a Büchi game in this case. More generally we plan to extend the framework we have introduced for fault diagnosis to control under dynamic partial observation and this will enable us to solve Problem 2.

Problem 3 is solved in doubly exponential time. Nevertheless to reduce the number of states of the most permissive observer, we point out that only *minimal* sets of events need to be observed. Indeed, if we can diagnose a system by observing only Σ from some point on, we surely can diagnose it using any superset $\Sigma' \supseteq \Sigma$. So far we keep all the sets that can be used to diagnose the system. We could possibly take advantage of the previous property using techniques described in [11].

REFERENCES

- [1] F. Cassez, S. Tripakis, and K. Altisen, "Synthesis of optimal dynamic observers for fault diagnosis of discrete-event systems," in *1st IEEE & IFIP Int. Symp. on Theoretical Aspects of Soft. Engineering (TASE'07)*. IEEE Computer Society, 2007, pp. 316–325.
- [2] —, "Sensor minimization problems with static or dynamic observers for fault diagnosis," in *7th Int. Conf. on Application of Concurrency to System Design (ACSD'07)*. IEEE Computer Society, 2007, pp. 90–99.
- [3] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, Jan. 1987.
- [4] J. Tsitsiklis, "On the control of discrete event dynamical systems," *Mathematics of Control, Signals and Systems*, vol. 2, no. 2, 1989.
- [5] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, Sept. 1995.
- [6] T.-S. Yoo and S. Lafortune, "On the computational complexity of some problems arising in partially-observed discrete event systems," in *American Control Conference (ACC'01)*, 2001, Arlington, VA.
- [7] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 8, Aug. 2001.
- [8] F. Cassez, S. Tripakis, and K. Altisen, "Sensor minimization problems with static or dynamic observers for fault diagnosis," IRCCyN/CNRS, 1 rue de la Noë, BP 92101, 44321 Nantes Cedex, France, Tech. Rep. RI-2007-1, Jan. 2007, available at <http://www.irccyn.fr/franck>.
- [9] R. Karp, "A characterization of the minimum mean cycle in a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1978.
- [10] U. Zwick and M. Paterson, "The complexity of mean payoff games on graphs," *Theoretical Computer Science*, vol. 158, no. 1–2, pp. 343–359, 1996.
- [11] L. Doyen, K. Chatterjee, T. Henzinger, and J.-F. Raskin, "Algorithms for omega-regular games with imperfect information," in *CSL: Computer Science Logic*, ser. Lecture Notes in Computer Science 4207. Springer, 2006, pp. 287–302.